华中科技大学

课 程 实 验 报 告

课程名称：C++程序设计

实验名称：面向过程的整型队列编程

院　　系 ：计算机科学与技术

专业班级 ：　CS2006

学　　号 ：　U202015471

姓　　名 ：　杨释钧

指导教师 ：　纪俊文

2021 年 10 月 20 日

目录

1 需求分析

1. 1 题目要求

整型队列是一种先进先出的存储结构，对其进行的操作通常包括：向队列尾部添加一个整型元素、从队列首部移除一个整型元素等。整型循环队列类型 Queue 及其操作函数采用非面向对象的 C 语言定义，请将完成上述操作的所有如下函数采用 C 语言编程，然后写一个 main 函数对队列的所有操作函数进行测试，请不要自己添加定义任何新的函数成员和数据成员。

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
struct Queue{
    int* const    elems;          //elems 申请内存用于存放队列的元素
     const    int    max;         //elems 申请的最大元素个数 max
    int     head, tail;           //队列头 head 和尾 tail，队空 head=tail;初始 head=tail=0
};
void initQueue(Queue *const p, int m); //初始化 p 指队列：最多申请 m 个元素
void initQueue(Queue *const p, const Queue&s); //用 s 深拷贝初始化 p 指队列
void initQueue(Queue *const p, Queue&&s); //用 s 移动初始化 p 指队列
int    number (const Queue *const p);   //返回 p 指队列的实际元素个数
int    size(const Queue *const p);              //返回 p 指队列申请的最大元素个数 max
Queue*const enter(Queue*const p, int e);   //将 e 入队列尾部，并返回 p
Queue*const leave(Queue*const p, int &e); //从队首出元素到 e，并返回 p
Queue*const assign(Queue*const p, const Queue&q); //深拷贝赋 s 给队列并返回 p
Queue*const assign(Queue*const p, Queue&&q); //移动赋 s 给队列并返回 p
char*print(const Queue *const p, char*s);//打印 p 指队列至 s 并返回 s
void destroyQueue (Queue *const p);     //销毁 p 指向的队列
```

编程时应采用 VS2019 开发，并将其编译模式设置为 X86 模式，其他需要注意的事项说明如下：

（1）用 initQueue(Queue *const p, int m)对 p 指向的队列初始化时， 为其 elems 分配 m 个整型元素内存，并初始化 max 为 m，以及初始化 head=tail=0。

（2）对于 initQueue(Queue *const p, const Queue& q)初始化，用已经存在的对象 q 深拷贝构造新对象*p 时，新对象*p 不能和对象 q 的 elems 共用同一块内存，新对象*p 的 elems 需要分配和 q 为 elems 分配的同样大小的内存，并且将已经存在 q 的 elems 的内容深拷贝至新分配的内存；新对象*p 的 max、head、tail 应设置成和已经存在的对象 s 相同。

（3）对于 initQueue(Queue *const p, Queue&& q)初始化，用已经存在的对象 q 移动构造新对象，新对象使用对象 q 为 elems 分配的内存快，并将其 max、head、tail 设置成和 s 的对应值相同，然后将 s 的 elems 设置为空表示内存已经移动给新对象，将 s 的 max、head、tail 设置为 0。

（4）对于 Queue*const assign(Queue*const p, const Queue&q)深拷贝赋值，用等号右边的对象 q 深拷贝赋值给等号左边的对象，等号左边的对象如果已经有内存则应先释放以避免内存泄漏，然后分配和对象 q 为 elems 分配的同样大小的内存，并且设置其 max、head、tail 和 q 的对应值相同。

（5）对于 Queue*const assign(Queue*const p, Queue&&q)移动赋值，若等号左边的对象为 elems 分配了内存，则先释放改内存一面内存泄漏，然后使用等号右边对象 q 为 elems 分配的内存，并将其 max、head、tail 设置成和对象 q 的对应值相同；对象 q 的 elems 设置为空指针以表示内存被移走给等号左边的对象，同时其 max、head、tail 均应设置为 0。

（6）队列应实现为循环队列，当队尾指针 tail 快要追上队首指针 head 时，即如果满足(tail+1)%max=head，则表示表示队列已满，故队列最多存放 max-1 个元素；而当 head=tail 时则表示队列为空。队列空取出元素或队列满放入元素均应抛出异常，并且保持其内部状态不变。

（7）打印队列时从队首打印至队尾，打印的元素之间以逗号分隔。

1.2 需求分析

用 C 语言实现以先进先出为特点的储存结构整型循环队列，实现队列所具有的操作。

整型循环队列类型 Queue 及其操作函数采用 C 语言定义，在 main 函数中对队列的所有操作函数进行测试库操作。

2 系统设计

2.1 概要设计

本实验主要通过指针和 malloc 函数分配内存以及强制类型转换方式实现对队列的初始化，入队出队列操作，及队列的删除释放等操作函数。需要注意的是我们要实现的队列是个循环队列，所以要对所有关于队首和队尾操作出仅需相应处理。

包括主函数模块，类定义模块，操作函数模块和测试库函数模块。模块间相应的层次结构为：主函数模块调用测试库函数，测试库函数调用操作函数模块，操作函数模块基于类定义模块设计。

对于 queue 的数据结构设计，首先有一个 elem 数组作为实际队列来储存数据，同时用两个 int 型 head, tail 来标记队列的队首和队尾的位置，用 int 型的 max 来标识队列的最大容量。

主要实现了如下操作：

1. 初始化 p 指队列：最多申请 m 个元素；
2. 用 s 深拷贝初始化 p 指队列；
3. 用 s 移动初始化 p 指队列；
4. 返回 p 指队列的实际元素个数；
5. 返回 p 指队列申请的最大元素个数 max；
6. 将 e 入队列尾部，并返回 p；
7. 从队首出元素到 e，并返回 p；
8. 深拷贝赋 s 给队列并返回 p；
9. 移动赋 s 给队列并返回 p；
10. 打印 p 指队列至 s 并返回 s；
11. 销毁 p 指向的队列；

循环队列如图 1. 1 所示。

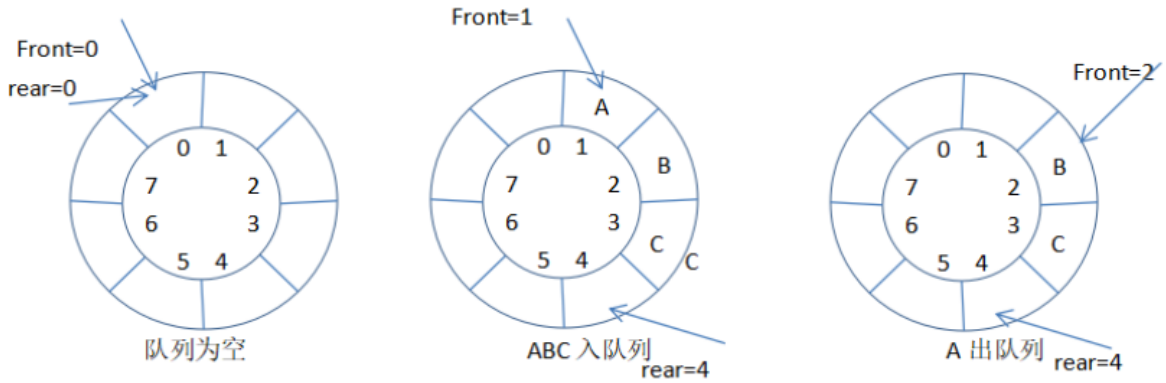

图 1. 1

类的定义以及每一种方法的声明如下：

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
struct Queue {
    int* const elems;      //elems申请内存用于存放队列的元素
    const int max;         //elems申请的最大元素个数max
    int head, tail;        //队列头head和尾tail，队空head=tail;初始head=tail=0
};
extern const char* TestQueue(int& s);
void initQueue(Queue* const p, int m);//初始化p指队列：最多申请m个元素
void initQueue(Queue* const p, const Queue& s); //用s深拷贝初始化p指队列
void initQueue(Queue* const p, Queue&& s); //用s移动初始化p指队列
int number(const Queue* const p);//返回p指队列的实际元素个数
int size(const Queue* const p);              //返回p指队列申请的最大元素个数max
Queue* const enter(Queue* const p, int e);  //将e入队列尾部，并返回p
Queue* const leave(Queue* const p, int& e); //从队首出元素到e，并返回p
Queue* const assign(Queue* const p, const Queue& q); //深拷贝赋s给队列并返回p
Queue* const assign(Queue* const p, Queue&& q); //移动赋s给队列并返回p
char* print(const Queue* const p, char* s);//打印p指队列至s并返回s
void destroyQueue(Queue* const p);       //销毁 p 指向的队列
```

2.2 详细设计

队列的数据成员包括指向用于存放队列的元素内存的整形指针 elem，申请的最大元素个数 max，指示队首与队尾的整形变量队列头 head 和尾 tail。相关操作及其说明如下：

(1)初始化队列：最多申请 m 个元素 void initQueue(Queue* const p, int m)：先为 elem 申请一块可以存放 m 个整形元素的内存，然后将 max 设置为 m，此时便完成了队列的创建，最后将 tail 与 head 设为 0，表示初始状态队列为空，队列初始化完成。

(2)用 q 深拷贝初始化队列 void initQueue(Queue* const p, const Queue& s)：深拷贝初始化要求新对象的 elems 需要分配和 q 为 elems 分配的同样大小的内存，并且将 q 的 elems 的内容深拷贝至新对象分配的内存；新对象的 max、head、tail 应设置成和 q 的对应值相同。因此先为 elems 分配一块大小为 q->max 大小的内存，然后将 max，head，tail 分别设为 q->max，q->head，q->tail，最后遍历 q 将 q 中所有元素赋给新对象的对应下标的元素。

(3)用 q 移动初始化队列 void initQueue(Queue* const p, Queue&& s)：移动初始化要求新对象接受使用对象 q 为 elems 分配的内存，并且新对象的 max、head、tail 应设置成和对象 q 的对应值相同；然后对象 q 的 elems 设置为空指针以表示内存被移走，同时其 max、head、tail 均应设置为 0。因此直接将 elems 指向 q->elems，然后将 max，head，tail 分别设为 q->max，q->head，q->tail，最后将 q->elems 设为 nullptr，将 q->max，q->head，q->tail 设为 0。其中由于 elems 和 max 都是只读类型的变量，所以修改其值时需要先取

地址，然后进行指针强制类型转换，最后再访问指针指向的内存即可。

(4)返回队列的实际元素个数 int　number(const Queue* const p)：由于队列时循环队列，故 tail 可能小于 head，因此队列的元素个数应为(tail+max-head)%max。考虑到 max 可能为 0，故只有再 max 不为 0 的时候返回(tail+max-head)%max，若为 0 则直接返回 0。

(5)返回队列申请的最大元素个数 virtual int size() const noexcept：直接返回 max 即可。

(6)将 e 入队列尾部，并返回当前队列 Queue* const enter(Queue* const p, int e)：入队前先要判满，循环队列满的条件为若再入一个元素则 tail 与 head 相等，即(tail+1)%max==head。若队列满则抛出异常，若队列未满则将元素放入 elem 中下标为 tail 的内存，最后改变 tail，根据循环队列的规律，tail 应该改变为(tail+1)%max。

(7)从队首出元素到 e，并返回当前队列 Queue* const leave(Queue* const p, int& e)：出队前先要判空，循环队列空的条件为 tail==head。若队列空则抛出异常，若队列不为空则将下标为 head 的值赋给 e，然后改变 head 的值，根据循环队列的规律，head 的值应该改变为(head+1)%max。

(8)深拷贝赋值并返回被赋值队列 Queue* const assign(Queue* const p, const Queue& q)：首先若传入的队列若就是自己，即 elem==q.elem，则没必要进行后面的操作，直接返回*this 即可。若 this 也被初始化过，即 elem 不为空，由于之后要对其分配新的内存使其指向的内存大小与 q.elem 相同，为防止内存泄漏，先释放 elem 的内存将其设为空，后面的操作与深拷贝初始化一致。

(9)移动赋值并返回被赋值队列：Queue* const assign(Queue* const p, Queue&& q)：首先若传入的队列若就是自己，即 elem==q.elem，则没必要进行后面的操作，直接返回*this 即可。若 this 也被初始化过，即 elem 不为空，由于之后要将其指向 q.elem，为防止内存泄漏，先释放 elem 的内存将其设为空，后面的操作与深拷贝初始化一致。

(10)打印队列至 s 并返回 char* print(const Queue* const p, char* s)：基本思路是使用双指针 i，j，i 用来表示当前读入的队列的元素的标号，j 用来表示当前的字符串数组中的下标，采用 sprintf 函数进行打印，注意最后为了方便，其实在数组的最开始多打印了一个逗号，需要用 strcpy 函数将该逗号消除。

(11)销毁当前队列 void destroyQueue(Queue* const p);：释放 elem 内存后将数据成员全部设为 0 即可。

3 软件开发与测试

3.1 软件开发

硬件环境：

- 处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
- 机带 RAM：16.0GB(15.4GB 可用)
- 系统类型：64 位操作系统，基于 x64 的处理器

开发环境：MSVC 2019 C++11，编译模式为 X86。 部分代码在 gcc， gdb 环境调试完成。

3.2 软件测试

测试结果如图 3.1 所示。

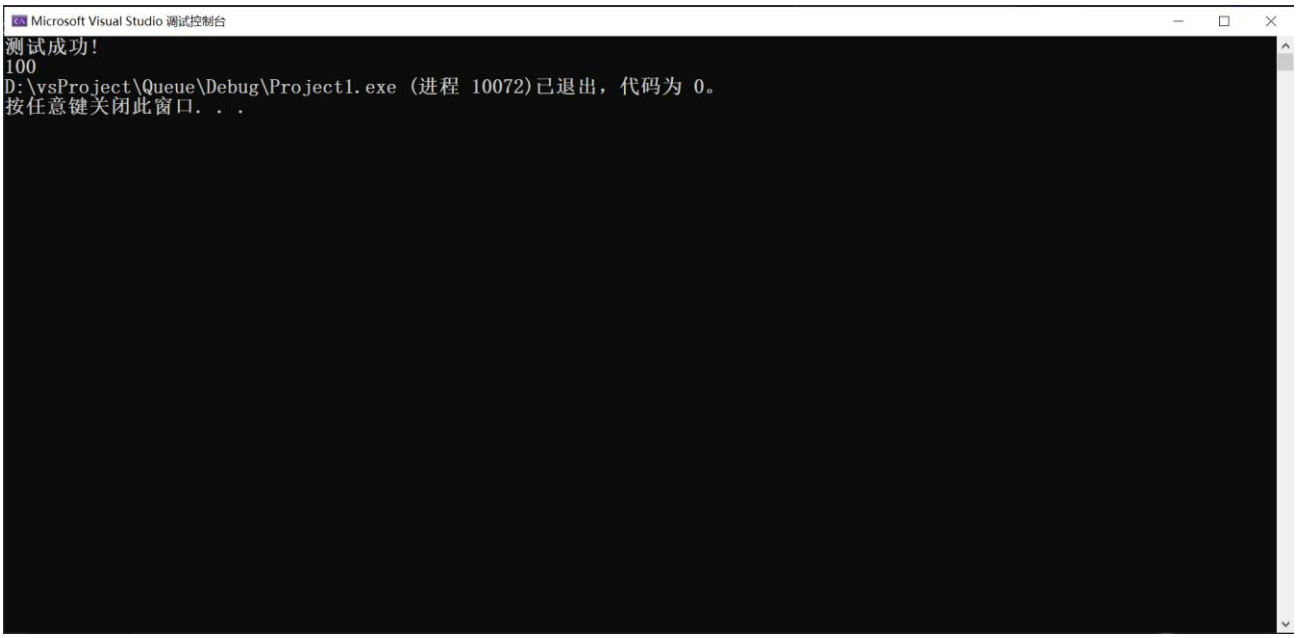

图 3.1

测试效果较好。

4 特点与不足

4.1 技术特点

采用面向过程的 C 语言实现了循环整形队列，整体代码较为简单易懂。

4.2 不足和改进的建议

队列为定长队列，泛用性差，可以改进为动态分配的内存的队列，即在队列满时调用 realloc 函数为队列重新分配一块更大的内存空间同时更新 max 的值，也可以使用链式队列，以提升空间利用率。

同时，由于本身受面向过程编程的影响，导致代码复用性并不好，当然作为一个从面向过程编程到面向对象编程过渡的实验，更多的是需要我们去体会两种编程模式的区别与联系，体会面向对象编程的优越性。

5 过程和体会

5.1 遇到的主要问题和解决方法

1. 由于过去并没有对 const 类型有过较多的理解与使用，所以在最开始不知道该如何对 const 类型的值进行赋值，最后在同学的帮助下解决了这个问题。

2. 由于本次实验是在面向过程编程和面向对象编程两种编程思想的衔接实验，所以原则上，initQueue 函数作为构造函数，是只能调用一次的，然而我最开始没有意识到这个问题，后来在老师的指导下才意识到了这一点。

5.2 课程设计的体会

本次实验加深了我对面向过程和面向对象编程两种编程思想的理解，同时让我对用 const 修饰的意义有了更进一步的理解，也了解到了对 const 修饰的变量进行修改的方法。

6 源码和说明

6.1 文件清单及其功能说明

提交文件中仅有 main.cpp 文件，相关函数的声明和定义全部放在这一个文件中，这也是这次实验我所欠缺的一点，即没有模块化进行编程的意识。

6.2 用户使用说明书

在 vs2019 中打开，并在本地运行。

6.3 源代码

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
struct Queue {
    int* const elems;      //elems申请内存用于存放队列的元素
    const int max;         //elems申请的最大元素个数max
    int head, tail;        //队列头head和尾tail，队空head=tail;初始head=tail=0
};
extern const char* TestQueue(int& s);
void initQueue(Queue* const p, int m);//初始化p指队列：最多申请m个元素
void initQueue(Queue* const p, const Queue& s); //用s深拷贝初始化p指队列
void initQueue(Queue* const p, Queue&& s); //用s移动初始化p指队列
int number(const Queue* const p);//返回p指队列的实际元素个数
int size(const Queue* const p);              //返回p指队列申请的最大元素个数max
Queue* const enter(Queue* const p, int e);  //将e入队列尾部，并返回p
Queue* const leave(Queue* const p, int& e); //从队首出元素到e，并返回p
Queue* const assign(Queue* const p, const Queue& q); //深拷贝赋s给队列并返回p
Queue* const assign(Queue* const p, Queue&& q); //移动赋s给队列并返回p
char* print(const Queue* const p, char* s);//打印p指队列至s并返回s
void destroyQueue(Queue* const p);       //销毁p指向的队列
int main()
{
    struct Queue* const p=(struct Queue*)malloc(sizeof(struct Queue));
   // for (int i = 1; i <= 5; i++)
     //    enter(p, i);
    int s = 0;
```

```
    const char* e = TestQueue(s);
    printf("%s", e);
    printf("\n%d", s);
}
void initQueue(Queue* const p, int m)
{
    *(int**)&(p->elems) = (int*)malloc(sizeof(int) * m);
    *(int*)&(p->max) = m;
    p->head = p->tail = 0;
}
void initQueue(Queue* const p, const Queue& s)
{
    *(int**)&(p->elems) = (int*)malloc(sizeof(int) * s.max);
    *(int*)&(p->max) = s.max;
    p->head = s.head;
    p->tail = s.tail;
    int i = s.head;
   // int j = 0;
    while (i != s.tail)
    {
        (*(int**)&(p->elems))[i]= s.elems[i];
        i = (i + 1) % s.max;
    }
}
void initQueue(Queue* const p, Queue&& s)
{
    *(int**)&(p->elems) = s.elems;
    p->head = s.head;
    *(int*)&p->max = s.max;
    p->tail = s.tail;
    //free(*(int**)&(s.elems));
    *(int**)&s.elems = NULL;
    s.head = 0;
    s.tail = 0;
    *(int*)&s.max = 0;
}
int number(const Queue* const p)
{
    if (p->elems == NULL)
        return 0;
    return (p->tail - p->head+p->max)%p->max;
}
int size(const Queue* const p)
{
    return p->max;
```

```
}
Queue* const enter(Queue* const p, int e)
{
    if ((p->tail + 1) % p->max == p->head)
    {
        throw("Queue is full!");
        return p;
    }
    (*(int**)&(p->elems))[p->tail] = e;
    p->tail = (p->tail + 1) % p->max;
    return p;
}
Queue* const leave(Queue* const p, int& e)
{
    if (p->head == p->tail)
    {
        throw("Queue is empty!");
        return p;
    }
    e = p->elems[p->head];
    p->head = (p->head + 1) % p->max;
    return p;
}
Queue* const assign(Queue* const p, const Queue& q)
{
    if (p == &q)
    {
        return p;
    }
    if (p != NULL)
    {
        free(p->elems);
        *(int**)&p->elems = NULL;
        *(int**)&p->elems = (int*)malloc(sizeof(int) * q.max);
    }
    for (int i = 0; i < q.max; i++)
        (*(int**)&p->elems)[i] = q.elems[i];
    *(int*)&p->max = q.max;
    p->head = q.head;
    p->tail = q.tail;
    return p;
}
Queue* const assign(Queue* const p, Queue&& q)
{
    if (p == & q)
```

```
    {
        return p;
    }
    if (p->elems != NULL) {
        free(p->elems);
        *(int**)&p->elems = NULL;
    }
    *(int**)&p->elems = q.elems;
    *(int*)&p->max = q.max;
    p->head = q.head;
    p->tail = q.tail;
    *(int**)&q.elems = NULL;
    *(int*)&q.max = q.head = q.tail = 0;
    return p;
}
char* print(const Queue* const p, char* s)
{
    int i = p->head;
    int x = 0;
    while (i != p->tail)
    {
        x = x+sprintf(s + x, ",%d", p->elems[i]);
        i = (i + 1) % p->max;
    }
    strcpy(s, s + 1);
    return s;
}
void destroyQueue(Queue* const p)
{
    free(p->elems);
    *(int**)&p->elems = NULL;
    *(int*)&p->max = 0;
    p->head = 0;
    p->tail = 0;
}
```